Appellant:	Joseph G. Laura	§	
		§	Group Art Unit: 2192
Serial No.:	10/723,967	§	
		§	Examiner: Wang, Ben C.
Filed:	November 26, 2003	§	
		§	Confirmation No. 9521
For:	APPLICATION MONITOR SYSTEM AND METHOD	§	
		§	
		§	

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

Dear Sirs:

This Appeal Brief is filed in support of the appeal in the above referenced application and is filed pursuant to the Notice of Appeal filed September 17, 2009 and the Notice of Panel Decision from Pre-Appeal Brief Review mailed November 20, 2009. Appellant authorizes all required fees under 37 C.F.R. § 1.17 to be charged to Deposit Account No. 21-0765, of Sprint Communications Company, L.P.

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

The real party in interest in the present application is the following party: Sprint Communications Company, L.P.

II. RELATED APPEALS AND INTERFERENCES

None.

III. STATUS OF CLAIMS

A. Total Number of Claims in the Application

The claims in the application are: 1-38

B. Status of All Claims in the Application

1. Claims canceled: None
2. Claims withdrawn from consideration but not canceled: None
3. Claims pending: 1-38
4. Claims allowed: None
5. Claims rejected: 1-38

C. Claims on Appeal

The claims on appeal are: 1-38

IV. STATUS OF AMENDMENTS

No amendments were filed after the March 17, 2009 Final Office Action, hereinafter "Final Office Action."

V. SUMMARY OF CLAIMED SUBJECT MATTER

The pending application relates to non-intrusively monitoring functioning computer software without encapsulating the software and without interfering with the performance of the software.[1] Application at 4, ¶ [0018], lines 5-8; Application at 6, ¶ [0022], lines 3-5; Application at 35, Abstract, lines 7-8. In particular, the pending application discloses a monitor attaching to and reading from memory blocks created by and written to during normal, real-time operation of an application without interfering with the normal operation of the application. Application at 4, ¶ [0018], line 5 through Application at 5, ¶ [0018], line 5; Application at 11, ¶ [0033], lines 4-5; Application at 16, ¶ [0044], lines 4-8; Application at 16, ¶ [0045], lines 2-4; Application at 18, ¶ [0049], lines 1-5; Application at 18, ¶ [0050], lines 1-8; Application at 19, ¶ [0052], lines 1-10; Figures 1 and 4.

Generally, it is useful to monitor software (e.g., applications, programs, subroutines, modules, etc.) during normal operations. Application at 1, ¶ [0005], line 1 through Application at 2, ¶ [0005], line 2. For example, conventional software applications may write intermediate information to certain internal variables, but the content of these intermediate variables is not output by the applications. *Id.* Consequently, this content is not accessible to conventional software test or development programs without changing or encapsulating the software application involved. *Id.* Therefore, it is more difficult for designers to develop, test or debug

[1] 37 C.F.R. § 41.37 (c)(1)(v) provides that the "[s]ummary of claimed subject matter . . . shall refer to the specification by page and line number." The instant application was presented in numbered page and numbered paragraph form. As such, the citations to the specification support for the claimed subject matter will be presented in the following form: Application at ___ (page number), ¶ [____] (paragraph number), lines ____ (lines within the corresponding paragraph). On the occasion when the pertinent paragraph is contained on multiple pages, the paragraph line numbering will begin anew on subsequent pages.

software applications using conventional techniques, without knowledge of the intermediate variables that were written as the software applications evolved. *Id.*

However, using one or more of the non-intrusive monitoring approaches disclosed in the pending application, a software monitor may attach to a block of memory of a software application. Application at 4, ¶ [0018], line 8 through Application at 5, ¶ [0018], line 2; Application at 11, ¶ [0033], lines 4-5; Application at 16, ¶ [0044], lines 4-6; Application at 16, ¶ [0045], lines 2-4; Application at 18, ¶ [0050], lines 1-8; and Figures 1 and 4. The software monitor may read application values that were written into the block of memory by the software application involved. Application at 4, ¶ [0018], line 8 through Application at 5, ¶ [0018], line 2; Application at 16, ¶ [0044], lines 4-6; Application at 16, ¶ [0045], lines 4-6; Application at 19, ¶ [0052], lines 1-10; and Figures 1 and 4. Thus, for example, a developer may view an application's intermediate variables in order to understand and correct the operation of the application during its testing phase, or a business analyst may access the internal variables of multiple software applications in order to optimize the interactions between the multiple applications involved. Application at 4, ¶ [0017], lines 2-5.

In a shared memory approach, the pending specification discloses that an application may be programmed or coded to create a shared memory and write values into variables contained within the shared memory whenever the application changes the value of those variables. Application at 4, ¶ [0018], lines 2-4; Application at 6, ¶ [0022], lines 1-3; elements 12, 22, 14, 24, 16, and 26 of Figure 1. The action of writing these values to the variables in the shared memory does not interfere with the normal operation of the application. Application at 6, ¶ [0022], lines 3-5. While the application is in real-time operation, a monitor module may be operable to read the values of the variables from the shared memory and communicate them to a

client. Application at 4, ¶ [0018], line 8 through Application at 5, ¶ [0018], line 2; Application at 5, ¶ [0019], lines 5-7; elements 18 and 20 of Figure 1. If the application does not write an internal variable to the shared memory, then this internal variable is not accessible by the monitor module and hence not accessible to the client. Application at 6, ¶ [0022], lines 11-13.

In some embodiments the monitor module and the client may be programmed in the Common Business Oriented Language (COBOL) programming language. Application at 8, ¶ [0026], lines 1-3. Unfortunately, COBOL may not provide support for shared memory. Application at 8, ¶ [0026], lines 3-4. In this case some embodiments may employ a technical layer which provides means for COBOL programs to avail themselves of certain features, including shared memory, which otherwise would not be accessible to these COBOL programs. Application at 8, ¶ [0026], lines 4-6. The technical layer may provide a shared memory routine. Application at 8, ¶ [0026], lines 6-8 and Figure 2. The shared memory routine may include an attach function to attach to an existing block of memory. Application at 11, ¶ [0033], lines 3-9.

In a memory attachment approach, it is not necessary for the application to mirror the values of internal variables by writing duplicate values in shared memory. Application at 18, ¶ [0049], lines 1-3. In this approach, the application does not need to be changed to enable operation of the system. Application at 18, ¶ [0049], lines 3-5. Typically, an application is loaded into memory of the computer system where it is being executed in various areas or sections, including an area reserved for variables defined in the application (e.g., a data section of an application's working memory). Application at 17, ¶ [0048], lines 3-10. The monitor module may be operable to attach to a memory area of the application and read the values of the variables from the memory area. Application at 16, ¶ [0044], lines 4-6 and Figure 4. Therefore,

in the memory attachment approach, all of the variables of the application are accessible to the monitor module. Application at 18, ¶ [0049], lines 9-10.

In the memory attachment approach, an address map may contain a listing of all variables names associated with the application and their address offset relative to the first instruction in the application executable file or relative to the beginning of the applications address space. Application at 18, ¶ [0051], line 3 - Application at 19, ¶ [0051], lines 1-7 and elements 116 and 120 of Figure 4. The monitor module or a client may use the address map to identify the address offset associated with a requested variable such that the monitor module may read the value at the offset in the memory area of the application. Application at 19, ¶ [0052], lines 1-10 and Figure 4.

Claim 1 is independent and is directed to a system for non-intrusively monitoring an application. Application at 4, ¶ [0018], lines 5-8; Application at 6, ¶ [0022], lines 3-5; Application at 35, Abstract, lines 7-8. The system comprises at least one application stored on a computer-readable medium, the at least one application creates a shared memory area and stores application values in the shared memory area. Figure 1, elements 12 and/or 22; Application at 4 ¶ [0018], lines 2-4; Application at 6, ¶ [0022], lines 1-3; ; Application at 9, ¶ [0028], lines 5-7; Application at 10, ¶ [0029], line 3 through Application at 10, ¶ [0030], line 6; and Application at 11, ¶ [0032], lines 5-6. The system comprises a first module stored on a computer-readable medium that shares and attaches to the shared memory area that is used by the at least one application during real-time operation. Figure 1, element 18; Figure 2, element 50; Application at 4, ¶ [0018], line 8 through Application at 5, ¶ [0018], line 5; Application at 11, ¶ [0033], lines 3-5. The first module reads application values from the shared memory area that have been stored in the shared memory area by the at least one application during real-time operation.

Application at 4, ¶ [0018], lines 5-9. The system also comprises a second module stored on a computer-readable medium in communication with the first module that requests the first module to read the application values. Figure 1, elements 20 and/or 32; Application at 5, ¶ [0019], lines 3-7 and 10-12; Application at 5, ¶ [0020], lines 1-5; Application at 8, ¶ [0026], lines 1-3. The second module receives the application values from the first module. Application at 5, ¶ [0019], lines 5-7; Application at 6, ¶ [0022], lines 9-11. The system further comprises a third module stored on a computer-readable medium in communication with the second module that displays the application values. Figure 1, element 28; Application at 5, ¶ [0019], lines 7-8; Application at 5, ¶ [0020], lines 1-2; Application at 6, ¶ [0022], lines 7-11; Application at 7, ¶ [0023], lines 1-8; and Application at 16, ¶ [0043], line 1.

Claim 2 depends on claim 1 and adds the limitation that the shared memory area is further defined as a shared memory of the application. Figure 1, elements 14 or 24; Application at 4 ¶ [0018], lines 2-4; Application at 6, ¶ [0022], lines 1-3; Application at 9, ¶ [0028], lines 5-7.

Claim 3 depends on claim 1 and adds the limitation that the first module is further configured to attach to the shared memory area used by the at least one application to read the application values. Application at 4, ¶ [0018], line 8 through Application at 5, ¶ [0018], line 5; Application at 11, ¶ [0033], lines 3-5.

Claim 4 depends on claim 1 and adds the limitation that the application values are further defined as at least one application variable and a value for the at least one application variable. Figure 1, elements 16 and 26; Application at 5, ¶ [0019], lines 1-8.

Claim 5 depends from claim 1 and adds the limitation that the first module is further configured to communicate the application values to the second module in hypertext markup

language format. Original claim 5 as recited in the Application at 25, lines 1-2; Application at 7, ¶ [0024], lines 1-4.

Claim 6 depends from claim 1 and adds the limitation that the third module is further defined as a graphical user interface. Figure 1, element 28; Application at 7, ¶ [0023], lines 1-8.

Claim 7 depends from claim 6 and adds the limitation that the graphical user interface is further configured to receive an input identifying the application values to be read and configured to request the application values identified to the first module, via the second module, and wherein the first module is configured to read the requested application values data from the shared memory area and return the application variables to the graphical user interface, via the second module. Application at 5, ¶ [0019], lines 1-8.

Claim 8 depends from claim 6 and adds the limitation that the graphical user interface is further configured to receive an input identifying requested application values to be displayed. Application at 5, ¶ [0019], lines 1-8.

Claim 9 depends from claim 1 and adds the limitation that the first module is further configured as a socket server and wherein the second module is further configured as a socket client such that the first and second modules communicate via a socket connection. Application at 7, ¶ [0025], line 1 through Application at 8, ¶ [0025], line 6.

Claim 10 depends from claim 1 and adds the limitation that the first module reads application values stored in the shared memory area by the at least one application while the at least one application is running. Application at 4, ¶ [0018], lines 6-9.

Claim 11 depends from claim 10 and adds the limitation that the first module reads application values stored in the memory area by the at least one application without interfering

with the operation of the at least one application. Application at 4, ¶ [0018], lines 5-8; Application at 6, ¶ [0022], lines 3-5; Application at 35, Abstract, lines 7-8.

Claim 12 is independent and is directed to a method of non-intrusively monitoring operation of an application. Application at 16, ¶ [0044], lines 1-2; Application at 35, Abstract, lines 7-8. The method comprises running an application in a real-time manner. Figure 4, elements 12 or 22; Application at 16, ¶ [0044], lines 7-8; and Application at 18, ¶ [0049], line 1. The method also comprises creating a memory area. Figure 4, elements 112 or 114; Application at 17, ¶ [0048], lines 1-10; and Application at 16, ¶ [0044], lines 4-6. The method also comprises generating, by the application, application values during operation of the application. Figure 4, elements 110; Application at 16, ¶ [0044], lines 7-8; and Application at 18, ¶ [0049], line 1. The method also comprises writing, by the application, the application values in the memory area during the operation of the application. Application at 16, ¶ [0044], lines 7-8; and Application at 18, ¶ [0049], line 1. The method also comprises reading, by a monitor, the memory area used by the application to obtain the application values. Figure 4, element 18; Application at 16, ¶ [0044], lines 4-6; Application at 18, ¶ [0050], lines 1-8; and Application at 19, ¶ [0052], lines 5-10. At least one of the application values is not output by the application. Application at 18, ¶ [0049], lines 1-11. The method also comprises displaying the application values read from the memory area. Figure 4, element 28 and/or 32; Application at 17, ¶ [0046], lines 3-4; Application at 20, ¶ [0053], lines 8-10.

Claim 13 depends from claim 12 and adds the limitation of requesting, by a client, application values from the monitor. Application at 18, ¶ [0050], lines 1-2; and Application at 19, ¶ [0052], lines 1-10. Claim 13 also adds the limitation of communicating the application

variables from the monitor to the client. Application at 16, ¶ [0044], lines 6-7; Application at 16, ¶ [0045], lines 4-6; Application at 19, ¶ [0052], lines 4-5 and 10; and Application at 20, ¶ [0055], lines 1-6.

Claim 14 depends from claim 13 and adds the limitation of requesting application values. Application at 16, ¶ [0044], lines 2-3; Application at 16, ¶ [0045], lines 1-2. Claim 14 also adds the limitation of running a plurality of applications in a real-time manner. Application at 16, ¶ [0044], lines 7-8; Application at 16, ¶ [0045], lines 3-4; and Application at 16 ¶ [0045], line 6 through Application at 17, ¶ [0045], line 1. Claim 14 also adds the limitation of generating application values stored in one or more memory areas during operation of the plurality of applications. Application at 16, ¶ [0044], lines 7-8; Application at 16, ¶ [0045], lines 3-4; and Application at 16 ¶ [0045], line 6 through Application at 17, ¶ [0045], line 1. Claim 14 also adds the limitation of reading the one or more memory areas used by the plurality of applications to obtain the application values. Application at 16; ¶ [0044], lines 4-6; and Application at 16, ¶ [0045], lines 4-6. Claim 14 also adds the limitation of displaying the requested application values. Application at 17, ¶ [0046], lines 3-4; Application at 20, ¶ [0053], lines 8-10.

Claim 15 depends from claim 14 and adds the limitation that the memory area is further defined as a block of memory and wherein the monitor reads at least some of the application variables stored in the block of memory. Figure 4, elements 112 and 114; Application at 16, ¶ [0044], lines 4-6; Application at 16, ¶ [0045], lines 4-6; and Application at 17, ¶ [0048], lines 1-10.

Claim 16 depends from claim 13 and adds the limitation of providing a memory manager and wherein the monitor registers with the memory manager to obtain a location of the memory

area used by the application to store the application values. Application at 18, ¶ [0050], lines 6-8; and original claim 16 in the Application at 28, lines 5-7.

Claim 17 depends from claim 13 and adds the limitation of generating new application values by the application stored in the memory area, at least one of the new application values defined as a new value for a variable of the application; requesting, by the client, that the monitor re-read the application values stored in the memory area; and re-reading, by the monitor, the memory area to obtain the new application values. Application at 19, ¶ [0053], line 1 through Application at 20, ¶ [0053], line 10; Application at 5, ¶ [0020], lines 1-5; Application at 6, ¶ [0022], line3; and original claim 17 in the Application at 28, lines 9-15.

Claim 18 depends from claim 17 and adds the limitation that the monitor reads the application values while the application is running. Application at 16, ¶ [0044], lines 4-8.

Claim 19 depends from claim 13 and adds the limitation that the monitor is configured as a socket server and wherein the client is configured as a socket client such that the communication between the monitor and client is via a socket connection. Application at 20, ¶ [0055], lines 1-6.

Claim 20 depends from claim 12 and adds the limitation that the application values are further defined as a variable of the application and a value of the variable. Figure 4, elements 110; and Application at 16, ¶ [0044], lines 2-6.

Claim 21 is independent and is directed to a system for non-intrusively monitoring variables during operation of an application. Application at 16, ¶ [0044], lines 1-2; and Application at 35, Abstract, lines 7-8. The system comprises a compile listing stored on a computer-readable medium having an address map with an offset associated with each of a plurality of variables of an application. Figure 4, elements 116 or 120; Application at 18, ¶

[0051], line 1 through Application at 19, ¶ [0051], line 7. The system also comprises a module stored on a computer-readable medium that performs reading of the compile listing and obtaining the offset of at least one of the plurality of variables of the application. Figure 4, elements 18 and/or 20; and Application at 19, ¶ [0052], lines 1-10. The module performs attaching to an address space used by the application during real-time operation to obtain a value for one or more of the plurality of variables written to the address space by the application during the real-time operation of the application using the offset. Application at 16, ¶ [0044], lines 2-8; Application at 18, ¶ [0050], lines 1-8; and Application at 19, ¶ [0052], lines 1-10.

Claim 22 depends from claim 21 and adds the limitation that the module is further configured to read the compile listing and convert at least one of the plurality of variables to the associated offset. Application at 19, ¶ [0052], lines 1-10.

Claim 23 depends from claim 21 and adds the limitation that the module is further configured to search the compile listing and display the plurality of variables of the application for selection by a user. Application at 17, ¶ [0046], lines 1-3; and Application at 19, ¶ [0052], lines 1-10.

Claim 24 depends from claim 23 and adds the limitation that the module is responsive to selection by the user of one of the plurality of variables to obtain the value for the selected one of the plurality of variables using the offset to locate the value of the variable in the address space. Application at 19, ¶ [0052], lines 1-10.

Claim 25 depends from claim 24 and adds the limitation that the module is further configured to display the selected one of the plurality of variables. Application at 17, ¶ [0046], lines 3-4; and Application at 19, ¶ [0052], lines 1-10.

Claim 26 depends from claim 21 and adds the limitation that the address space is further defined as a memory space and wherein the module attaches, using a socket layer, to the memory space used by the application. Application at 17, ¶ [0048], lines 1-10; Application at 20, ¶ [0055], lines 1-6.

Claim 27 depends from claim 26 and adds the limitation that the module attaches, using the offset, to the memory space used by the application via an operating system service. Application at 18, ¶ [0050], lines 2-3; Application at 20, ¶ [0055], lines 1-6.

Claim 28 depends from claim 21 and adds the limitation that the monitor is further configured, using the compile listing, to query the address map for one or more of the plurality of variables of the application. Application at 18, ¶ [0051], line 1 through Application at 19, ¶ [0051], line 7; and Application at 19, ¶ [0052], lines 1-10.

Claim 29 depends from claim 21 and adds the limitation that the module is further defined as a subtask of the operating system. Original claim 29 in the Application at 31, lines 11-12.

Claim 30 depends from claim 21 and adds the limitation that the module is further configured to attach to the memory space where the application is operating and overwrite the value for one or more of the plurality of variables using the offset. Original claim 30 in the Application at 31, lines 14-16.

Claim 31 depends from claim 21 and adds the limitation that the module comprises a reader component configured to perform reading the compile listing and further configured to perform converting at least one of the plurality of variables of the application to the associated offset. Figure 4, element 20; and Application at 19, ¶ [0052], lines 1-10. Claim 31 adds the

limitation that the module also comprises a search component that performs receiving the associated offset of the at least one of the plurality of variables from the reader component, the search component configured to perform attaching to the application and further operable to locate the value of the at least one of the plurality of variables using the offset. Figure 4, element 18; and Application at 19, ¶ [0052], lines 1-10.

Claim 32 depends from claim 21 and adds the limitation of a display component operably coupled to the module to perform receiving the value for the one or more of the plurality of variables, the display component configured to perform displaying the value. Figure 4, elements 28 and/or 32; Application at 17, ¶ [0046], lines 3-4; and Application at 19, ¶ [0053], line 1 through Application at 20, ¶ [0053], line 10.

Claim 33 depends from claim 32 and adds the limitation that the display component is configured to employ the value to display a heartbeat. Application at 19, ¶ [0053], line 1 through Application at 20, ¶ [0053], line 10.

Claim 34 depends from claim 32 and adds the limitation that the display component is configured to employ the value to display as a percentage complete. Original claim 34 in the Application at 32, lines 12-13.

Claim 35 is independent and is directed to a system for non-intrusively monitoring COBOL application values. Application at 4, ¶ [0018], lines 5-8; Application at 6, ¶ [0022], lines 3-5; Application at 8, ¶ [0026], lines 1-3; Application at 35, Abstract, lines 7-8. The system comprises a COBOL program stored on a computer-readable medium that creates a shared memory area through a technical layer, generates program values, and stores the program values in the shared memory area during real-time operation of the COBOL program. Figure 1, elements 12 and/or 22; Application at 4 ¶ [0018], lines 2-8; Application at 6, ¶ [0022], lines 1-3;

Application at 8, ¶ [0026], lines 1-3 and 6-8; Application at 9, ¶ [0028], lines 5-7; Application at 10, ¶ [0029], line 3 through Application at 10, ¶ [0030], line 6; and Application at 11, ¶ [0032], lines 5-6. The system also comprises a COBOL monitor module stored on a computer-readable medium that shares the shared memory area with the COBOL program through the technical layer, and the COBOL monitor module reads the program values stored in the shared memory area by the COBOL program during real-time operation of the COBOL program. Figure 1, element 18; Figure 2, element 50; Application at 4, ¶ [0018], line 8 through Application at 5, ¶ [0018], line 5; Application at 11, ¶ [0033], lines 3-5.

Claim 36 depends from claim 35 and adds the limitation of a second COBOL program configured to generate second program values and store the program values in the shared memory area during real-time operation of the second COBOL program, and wherein the COBOL monitor module is further configured to read the second program values stored in the shared memory area by the second COBOL program. Figure 1, elements 12 or 22; Application at 4, ¶ [0018], line 8 through Application at 5, ¶ [0018], line 5; and Application at 8, ¶ [0026], lines 1-3.

Claim 37 depends from claim 35 and adds the limitation of a second memory area. Figure 1, elements 14 or 24. Claim 37 also adds the limitation of a second COBOL program configured to generate second program values and store the program values in the second memory area during real-time operation of the second COBOL program, and wherein the COBOL monitor module is further configured to read the second program values stored in the second memory area by the second COBOL program. Figure 1, elements 12 or 22; Application at 4, ¶ [0018], line 8 through Application at 5, ¶ [0018], line 5; and Application at 8, ¶ [0026], lines 1-3.

Claim 38 depends from claim 34 and adds the limitation of a user interface configured to monitor and display the application values. Figure 1, element 28 and/or 32; and Application at 5, ¶ [0019], lines 7-8. Claim 38 also adds the limitation of a client application in communication with the user interface and the COBOL monitor module, the client application configured to request the program variables of the COBOL program from the COBOL monitor module and provide the program variables to the user interface for display via the user interface responsive to a request from the user interface. Figure 1, elements 20 and/or 32; and Application at 5, ¶ [0019], lines 3-7.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

1. Whether claims 21-24, 26, 28, 30, and 35-37 are anticipated under 35 U.S.C. § 102(3) by Nace et al. (U.S. Patent Application Publication 2004/0268363) (hereinafter "*Nace*").
2. Whether claims 1-4, 6-20, and 38 are unpatentable under 35 U.S.C. § 103(a) as obvious over *Nace* in view of Sridharan et al. (*On Building Non-Intrusive Performance Instrumentation Blocks for CORBA-based Distributed Systems*, March 2000, IEEE) (hereinafter "*Sridharan*").
3. Whether claim 5 is unpatentable under 35 U.S.C. § 103(a) as obvious over *Nace* in view of *Sridharan* and further in view of Hiroshi Kashima (*An Approach for Constructing Web Enterprise Systems on Distributed Objects*, January 2000, IBM) (hereinafter "*Kashima*").
4. Whether claims 25, 31, and 32 are unpatentable under 35 U.S.C. § 103(a) as obvious over *Nace* in view of Jie Tao et al. (*Visualizing the Memory Access Behavior of Shared Memory Applications on NUMA Architectures*, Springer-Verlag Berlin Heidelberg 2001, pp. 861-870) (hereinafter "*Tao-2*").
5. Whether claims 27 and 29 are unpatentable under 35 U.S.C. § 103(a) as obvious over *Nace* in view of Huang et al. (*Operating System Support for Flexible Coherence in Distributed Shared Memory*, 1996, IEEE) (hereinafter "*Huang*").
6. Whether claims 33 and 34 are unpatentable under 35 U.S.C. § 103(a) as obvious over *Nace* in view of *Tao-2* and further in view of Tao et al. (*Understanding the Behavior of Shared Memory Applications Using the SMiLE Monitoring Framework*, March 2000, IEEE)(hereinafter "*Tao-1*").

VII. ARGUMENT

The Examiner has failed to establish a *prima facie* case of anticipation with respect to claims 21-24, 26, 28, 30, and 35-37. According to MPEP § 2131, "[a] claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." Appellant respectfully submits that *Nace* does not explicitly or inherently disclose each and every element of claims 21-24, 26, 28, 30, and 35-37.

The Examiner has also failed to establish a *prima facie* case of obviousness with respect to claims 1-20, 25, 27, 29, 31-34, and 38. As noted by the United States Supreme Court in *Graham v. John Deere Co. of Kansas City*, an obviousness determination begins with a finding that **"the prior art as a whole in one form or another contains all" of the elements of the claimed invention**. *See Graham v. John Deere Co. of Kansas City*, 383 U.S. 1, 22 (U.S. 1966). Appellant respectfully submits that *Nace* alone or in combination with the other applied art does not contain all of the elements of claims 1-20, 25, 27, 29, 31-34, and 38.

More specifically, *Nace* does not disclose an application that *creates* a shared memory area. *Nace* discloses an interprocess communications platform that enables interprocess communications through blocks of a shared memory space. *Nace*, Abstract. The interprocess communications platform includes a shared memory space accessible by a plurality of processes of an arbitrary number of applications. *Nace*, Figure 2 and ¶ [0015]. Each of the processes may have at least one memory block in a set of memory blocks of the shared memory space uniquely assigned or mapped to it. *Nace*, ¶ [0016]. While *Nace* may disclose that memory blocks may be assigned to a process, *Nace* does not disclose that the processes create the shared memory or memory blocks within the shared memory. Rather, *Nace* discloses that a new or newly registered memory block within the set of memory blocks in the shared memory may be secured

or generated *by administrative memory 106* or other resources. *Nace*, ¶ [0031]. Further, *Nace* discloses that the administrative memory 106 may be generated by the communications engine 108. *Nace*, ¶ [0018]. Accordingly, *Nace* does not disclose an application that *creates* a shared memory area.

Also, *Nace* does not disclose a COBOL program and a COBOL monitor module. Nace discloses that although specific languages, routines or other code characteristics are illustrated in the computer code for generating a communications session as illustrated in Fig. 3, different types of code or instructions may be used. While *Nace* may disclose that different code *may* be used, *Nace* does not provide any explicit or inherent disclosure that the different code may be COBOL. Appellant notes, "'Inherency, however, may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient.' " *In re Robertson*, 169 F.3d 743, 745, 49 USPQ2d 1949, 1950-51 (Fed. Cir. 1999) (citations omitted). Furthermore, as noted in paragraph [0026] of the pending application, COBOL may not provide support for sockets, shared memory, or other POSIX functionality that may be needed to enable the interprocess communication disclosed by *Nace*. Accordingly, even with the suggestion of *Nace* that different code *may* be used, one of ordinary skill in the art would not have looked to use COBOL code. As disclosed in the pending application, shared memory and socket functionality are enabled for the COBOL applications through a technical layer that comprises shared memory and socket routines. Figs. 2 and 3 and paragraphs [0026]-[0042].

Moreover, *Nace* does not disclose a monitor that reads application values from a memory area created by an application, wherein at least one of the application values is not output by the application. *Nace* discloses that when a process is initiated it may request the creation or

allocation of a memory block within the set of memory blocks in the shared memory space. *Nace*, ¶ [0021]. Other data related to the newly-generated shared memory block and its associated process may be loaded or stored in the administrative memory. *Nace*, ¶s [0022] and [0031]. This other data may include file handles, variables, address offset or other memory mapping data and other information related to the requesting process. *Nace*, ¶ [0022]. *Nace* discloses that once the configuration is complete, the requesting process may operate on its corresponding memory block to read and write data, such as variables, arrays, tables, or other content. *Id.* Accordingly, it is clear that this other data that is loaded in the administrative memory is data that is output by the corresponding process.

In fact, *Nace* explicitly discloses that the administrative memory tracks indicators of memory areas *populated by* (i.e. output by) individual processes. *Nace*, Abstract. Moreover, *Nace* discloses that the field of the invention is to enable applications executing in a shared memory space to *exchange* data. *Nace*, ¶ [0003]. In order to enable the exchange of data between processes, *Nace* discloses that an initiating process must populate (i.e. output to) a memory block with data to be communicated to a destination process. *Nace*, ¶ [0035]. See also Fig. 5 and paragraphs [0034]-[0037] and [0040]. Accordingly, *Nace* does not disclose a monitor that reads application values from a memory area created by an application, wherein at least one of the application values is not output by the application.

As discussed above, the memory attachment approach disclosed by the pending application enables the monitor module to attach to and read from areas of an application that are loaded into memory, such as an area reserved for variables defined in the application (e.g., a data section of an application's working memory). Figure 4; Application at 16, ¶ [0044], lines 4-6; and Application at 17, ¶ [0048], lines 3-10. Because the monitor module is attached to an area of

memory reserved for variables defined in the application, it is not necessary for the application to mirror the values of internal variables by writing duplicate values in shared memory. Also, the application does not need to be changed to enable operation of the system. Therefore, all of the variables of the application are accessible to the monitor module, including internal variables of an application that are not output by the application. Application at 18, ¶ [0049], lines 1-5; and Application at 18, ¶ [0049], lines 9-10.

Moreover, *Nace* does not disclose a compile listing having an address map with an offset associated with each of a plurality of variables of an application and a monitor that obtains a value for one or more of the plurality of variables using the offset. *Nace* may disclose that other data related to a newly-generated shared memory block may be loaded into administrative memory, such as file handles, variables, address offset or other memory mapping data and other information related to an initiating process. *Nace*, ¶ [0022]. While *Nace* may disclose this other data may include address offset or other memory mapping data, *Nace* does not disclose that the memory mapping data is an address map with an offset associated with each of a plurality of variables of an application. Also, *Nace* does not disclose that memory mapping data is part of a *compile listing*. As disclosed in paragraph [0052] of the pending application, the claimed address map may be generated when compiling and linking applications. Accordingly, the claimed compile listing having the address map provides the monitor module with detailed information regarding the mapping of variables within reserved areas of an application's working memory. Further, *Nace* does not disclose using the address offset or other memory mapping data stored in the administrative memory to obtain a value of one or more variables of an application using the address offset.

These and other distinctions will be addressed in more detail below with reference to the specific language of the claims. While only *Nace* was addressed above, Appellant submits that the other applied art does not cure the deficiencies of *Nace* discussed above. The combination of references used in the obviousness rejections when considered as a whole will also be addressed in more detail below.

A. 35 U.S.C. § 102 Rejections – Claims 21-24, 26, 28, 30, and 35-37 rejected over *Nace*.

1. Claims 21-24, 26, 28, and 30 are not anticipated by *Nace*.

a. Claims 21-24, 26, 28, and 30 were wrongly rejected because *Nace* does not expressly or inherently disclose a compile listing having an address map with an offset associated with each of a plurality of variables of an application.

Claim 21 recites, "a compile listing stored on a computer-readable medium having an address map with an offset associated with each of a plurality of variables of an application."

With reference to the above claimed features, the Final Office Action stated, "Fig. 2, element - Mapping; element 122 - API to memory managers; element 118a - Memory Manager; [0008], Lines 13-15 - A communications engine may map a logical address for the memory block to physical memory within its address space on a dynamic basis; [0022] - …variables, address offset or other memory mapping data and other information related to the requesting process …to read and write data, such as variable, arrays, tables …; [0039] - … by computation of the offset or other known relation to its associated buddy block or segment." Final Office Action, page 3, emphasis in original.

While the above citations of *Nace* may generally disclose memory mapping, *Nace* does not disclose any such memory mapping data is part of a **compile listing**, as claimed (e.g., "a

compile listing …having an address map"). As disclosed in paragraph [0052] of the pending application, the claimed address map may be generated when compiling and linking applications. Accordingly, the claimed compile listing having the address map provides the monitor module with detailed information regarding the mapping of variables within reserved areas of an applications working memory. With the knowledge of the mapping of variables within reserved memory areas of an application, the monitor module may monitor the application by reading the values of the variables without having to modify the application. Also, all of the variables of the application are accessible to the monitor module, including internal variables of an application that are not output by the application. Application at 18, ¶ [0049], lines 1-5; and Application at 18, ¶ [0049], lines 9-10.

Also, while *Nace* may disclose that an administrative memory may store address offset or other memory mapping data or other data related to a requesting process, *Nace* does not explicitly or inherently disclose that such data includes an address map with an offset associated with each of a plurality of variables of an application, as claimed. Further, based on the disclosure of *Nace* it is not inherent that the address offset or other memory mapping data would include the claimed address map. For example, the address offset or other memory mapping data simply may be an offset within the shared memory space at which a particular block of shared memory begins.

Moreover, mapping a logical address to a physical address does not provide explicit or inherent disclosure of a compile listing having an address map with an offset associated with each of a plurality of variables of an application. Similarly, disclosure of computation of an offset between a shared memory block and its associated buddy block (e.g., computation of an offset between two shared memory blocks) does not provide explicit or inherent disclosure of a

compile listing having an address map with an offset associated with each of a plurality of variables of an application.

Accordingly, for the reasons stated above, *Nace* does not explicitly or inherently disclose a compile listing stored on a computer-readable medium having an address map with an offset associated with each of a plurality of variables of an application.

Claims 22-24, 26, 28, and 30 depend from claim 21. Therefore, the arguments presented above in section VII.A.1.a. are repeated for these claims.

b. Claims 21-24, 26, 28, and 30 were wrongly rejected because *Nace* does not expressly or inherently disclose a module performs reading of the compile listing and obtaining the offset of at least one of the plurality of variables to obtain a value for one or more of the plurality of variables using the offset.

Claim 21 recites, "a module stored on a computer-readable medium that performs reading of the compile listing and obtaining the offset of at least one of the plurality of variables of the application, the module performs attaching to an address space used by the application during real-time operation to obtain a value for one or more of the plurality of variables written to the address space by the application during the real-time operation of the application using the offset."

As noted above, *Nace* does not disclose the compile listing having the address map and accordingly cannot disclose a module that reads the compile listing to obtain the offset of at least one variable, as claimed. Nevertheless, the Final Office Action again relied on much of the disclosure of *Nace* cited above. However, with regard to a module obtaining a value of one or more variables using the offset of the offset map, the Final Office Action references paragraph [0022].

In particular, the Final Office Action states, "e.g., [0022] - ...variable, address offset or other memory mapping data and other information related to the requesting process ... to read and write data, such as variable, arrays, tables ..." Final Office Action, page 4. However, Appellant submits that the above citation is taken out of context. In the paragraph [0021] *Nace* begins to disclose the procedure by which a process is initiated. As part of this initiation procedure, *Nace* discloses in paragraph [0022] that other data may be loaded into an administrative memory, including file handles, variables, address offset or other memory mapping data and other information related to the requesting process (i.e., the process that is being initiated). In a separate sentence and concluding the procedure for initiating a process, *Nace* states, "When configuration is complete, the requesting process ...may communicate with and operate on its corresponding memory block ...to read and write data, such as variables, arrays, tables or other content." Therefore, contrary to what one might think upon reading the citation from the Final Office Action, *Nace* does not actually disclose in paragraph [0022] to use the address offset or other memory mapping data to read and write data to a memory block.

Accordingly, *Nace* does not explicitly or inherently disclose a module performs reading of the compile listing and obtaining the offset of at least one of the plurality of variables to obtain a value for one or more of the plurality of variables using the offset.

Claims 22-24, 26, 28, and 30 depend from claim 21. Therefore, the arguments presented above in section VII.A.1.b. are repeated for these claims.

c. Claims 23 and 24 were wrongly rejected because *Nace* does not expressly or inherently disclose the module is further configured to search the compile listing and display the plurality of variables of the application for selection by a user.

Claim 23 recites, "wherein the module is further configured to search the compile listing and display the plurality of variables of the application for selection by a user." Claim 24 depends from claim 23 and further recites, "wherein the module is responsive to selection by the user of one of the plurality of variables to obtain the value for the selected one of the plurality of variables using the offset to locate the value of the variable in the address space."

The Final Office Action relied on many of the same citations of *Nace* discussed above relating to mapping a logical address to physical memory, storing address offset or other memory mapping data, and computing offsets between buddy blocks. However, as noted above, *Nace* does not disclose the compile listing having the address map and accordingly cannot disclose searching the compile listing and displaying the plurality of variables, as claimed. Further, none of the portions of *Nace* cited by the Final Office Action disclose searching anything and displaying the results for selection by a user. Also, none of the portions of *Nace* cited by the Final Office Action disclose obtaining a value of a variable responsive to user selection, as claimed. More particularly, a text search for the terms "search," "display," "user," and "select" did not produce any relevant results in *Nace*.

Accordingly, *Nace* does not explicitly or inherently disclose the module is further configured to search the compile listing and display the plurality of variables of the application for selection by a user. Nor does *Nace* disclose the module is responsive to selection by the user of one of the plurality of variables to obtain the value for the selected one of the plurality of variables using the offset to locate the value of the variable in the address space.

2. Claims 35-37 are not anticipated by *Nace*.

a. Claims 35-37 were wrongly rejected because *Nace* does not expressly or inherently disclose a COBOL program and a COBOL monitor module.

Claim 35 recites, "a COBOL program …; and a COBOL monitor module."

With regard to the COBOL programming language, the Final Office Action simply cites paragraph [0028] of *Nace*, which discloses with reference to the exemplary computer code of Fig. 3, "Although specific languages, routines or other code characteristics are illustrated, it will be appreciated that different types of code or instruction may be used." While *Nace* may disclose that different code **may** be used, *Nace* does not provide any explicit or inherent disclosure that the different code may be COBOL. Appellant notes, "'Inherency, however, may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient.' " *In re Robertson*, 169 F.3d 743, 745, 49 USPQ2d 1949, 1950-51 (Fed. Cir. 1999) (citations omitted). Furthermore, as noted in paragraph [0026] of the pending application, COBOL may not provide support for sockets, shared memory, or other POSIX functionality that may be needed to enable the interprocess communication disclosed by *Nace*. Accordingly, even with the suggestion of *Nace* that different code *may* be used, one of ordinary skill in the art would not have looked to use COBOL code. As disclosed in the pending application, shared memory and socket functionality are enabled for the COBOL applications through a technical layer that comprises shared memory and socket routines. Figs. 2 and 3 and paragraphs [0026]-[0042].

Claims 36 and 37 depend from claim 35. Therefore, the arguments presented above in section VII.A.2.a. are repeated for these claims.

b. Claims 35-37 were wrongly rejected because *Nace* does not expressly or inherently disclose an application that creates a shared memory area.

Claim 35 recites, "a COBOL program stored on a computer-readable medium that creates a shared memory area."

Nace discloses an interprocess communications platform that enables interprocess communications through blocks of a shared memory space. *Nace*, Abstract. The interprocess communications platform includes a shared memory space accessible by a plurality of processes of an arbitrary number of applications. *Nace*, Figure 2 and ¶ [0015]. Each of the processes may have at least one memory block in a set of memory blocks of the shared memory space uniquely assigned or mapped to it. *Nace*, ¶ [0016]. While *Nace* may disclose that memory blocks may be assigned to a process, *Nace* does not disclose that the processes create the shared memory or memory blocks within the shared memory. Rather, *Nace* discloses that a new or newly registered memory block within the set of memory blocks in the shared memory may be secured or generated *by administrative memory 106* or other resources. *Nace*, ¶ [0031]. Further, *Nace* discloses that the administrative memory 106 may be generated by the communications engine 108. *Nace*, ¶ [0018]. Accordingly, *Nace* does not disclose an application that *creates* a shared memory area. Further, *Nace* does not explicitly or inherently disclose that a COBOL program creates a shared memory area *through a technical layer*, as claimed.

Claims 36 and 37 depend from claim 35. Therefore, the arguments presented above in section VII.A.2.b. are repeated for these claims.

B. 35 U.S.C. § 103 Rejections – Claims 1-4, 6-20, and 38 are rejected over *Nace* in view of *Sridharan*.

1. *Nace* in view of *Sridharan* do not render claims 1-4, and 6-11 obvious.

a. Claims 1-4 and 6-11 were wrongly rejected because *Nace* in view of *Sridharan* do not teach or suggest at least one application that creates a shared memory area and stores application values in the shared memory area.

Claim 1 recites, "at least one application stored on a computer-readable medium, the at least one application creates a shared memory area and stores application values in the shared

memory area." Appellant notes that a similar feature was addressed above with reference to the rejection of claim 35. Therefore, the arguments presented above in section VII.A.2.b. are repeated for these claims. The Final Office Action did not rely on the disclosure of *Sridharan* to teach the above features. However, Appellant submits that *Sridharan* does not cure the deficiencies in *Nace* noted above.

Claims 2-4 and 6-11 depend from claim 1. Therefore, the arguments presented in section VII.B.1.a. are repeated for these claims.

b. Claims 1-4 and 6-11 were wrongly rejected because *Nace* in view of *Sridharan* do not teach or suggest a third module in communication with the second module that displays the application values.

Claim 1 recites, "a third module stored on a computer-readable medium in communication with the second module that displays the application values." The Final Office Action states, "Nace …does not explicitly disclose a third module stored on a computer-readable medium in communication with the second module, that displays the application values." The Final Office Action goes on to rely on the disclosure of *Sridharan* and states, "Fig. 3 - System T and Performance Instrumentation, element of "PM GUI"; P. 3, last Par. - A GUI was implemented to display the various performance data, to set the filtering information and turn on and off the monitoring mechanism."

Sridharan is directed to a proposal for a non-intrusive framework for collecting performance statistics of Common Object Request Broker Architecture (CORBA)-based distributed systems. *Sridharan* describes an application of its non-intrusive framework to a large telecommunications system and the experimental results obtained. Specifically, *Sridharan's* study describes use of a Visibroker Object Request Broker (ORB) to deploy servers in the telecommunication system involved. The Visibroker ORB loads a performance instrumentation

module together with a server into a single address space. According to *Sridharan*, sharing of the address space by an instance of the performance instrumentation module helps in collecting server specific information such as the CPU time, memory utilization, and the number of ORB threads launched. In other words, *Sridharan* generally describes a server and performance instrumentation module that share an address space so the performance instrumentation module can retrieve performance details for the particular server involved. As noted by the Final Office Action, *Sridharan* may disclose a GUI for displaying the performance data of the server. However, the performance data does not include specific values of variables of an application that is being monitored. Accordingly, even if *Nace* and *Sridharan* were combined, the resultant combination would not disclose displaying application values read by the monitor module, as claimed.

Claims 2-4 and 6-11 depend from claim 1. Therefore, the arguments presented in section VII.B.1.b. are repeated for these claims.

2. *Nace* in view of *Sridharan* do not render claims 12-20 obvious.

a. Claims 12-20 were wrongly rejected because *Nace* in view of *Sridharan* do not teach or suggest reading, by a monitor, the memory area used by the application to obtain application values, wherein at least one of the application values is not output by the application.

Claim 12 recites, "reading, by a monitor, the memory area used by the application to obtain application values, wherein at least one of the application values is not output by the application."

In addressing the above features of claim 12, the Final Office Action relies on the disclosure of *Nace* and states, "Abstract, Lines 6-11 - … via an administrative memory space which tracks pointers, handles and other indicators of memory area populated by individual

processes. When one process requests access to a variable, pointer or other data generated by another process, the request is mediated by the communications engine." Final Office Action, page 18, emphasis in original.

Nace discloses that when a process is initiated it may request the creation or allocation of a memory block within the set of memory blocks in the shared memory space. *Nace*, ¶ [0021]. Other data related to the newly-generated shared memory block and its associated process may be loaded or stored in the administrative memory. *Nace*, ¶s [0022] and [0031]. This other data may include file handles, variables, address offset or other memory mapping data and other information related to the requesting process. *Nace*, ¶ [0022]. *Nace* discloses that once the configuration is complete, the requesting process may operate on its corresponding memory block to read and write data, such as variables, arrays, tables, or other content. *Id.* Accordingly, it is clear that this other data that is loaded in the administrative memory is data that is output by the corresponding process.

In fact, as quoted above *Nace* explicitly discloses that the administrative memory tracks indicators of memory areas *populated by* (i.e. output by) individual processes. *Nace*, Abstract. Moreover, *Nace* discloses that the field of the invention is to enable applications executing in a shared memory space to *exchange* data. *Nace*, ¶ [0003]. In order to enable the exchange of data between processes, *Nace* discloses that an initiating process must populate (i.e. output to) a memory block with data to be communicated to a destination process. *Nace*, ¶ [0035]. See also Fig. 5 and paragraphs [0034]-[0037] and [0040]. Accordingly, *Nace* does not disclose a monitor that reads application values from a memory area created by an application, wherein at least one of the application values is not output by the application.

As discussed above, the memory attachment approach disclosed by the pending application enables the monitor module to attach to and read from areas of an application that are loaded into memory, such as an area reserved for variables defined in the application (e.g., a data section of an application's working memory). Figure 4; Application at 16, ¶ [0044], lines 4-6; and Application at 17, ¶ [0048], lines 3-10. Because the monitor module is attached to an area of memory reserved for variables defined in the application, it is not necessary for the application to mirror the values of internal variables by writing duplicate values in shared memory. Also, the application does not need to be changed to enable operation of the system. Therefore, all of the variables of the application are accessible to the monitor module, including internal variables of an application that are not output by the application. Application at 18, ¶ [0049], lines 1-5; and Application at 18, ¶ [0049], lines 9-10.

Claims 13-20 depend from claim 12. Therefore, the arguments presented in section VII.B.2.a. are repeated for these claims.

3. *Nace* in view of *Sridharan* do not render claim 38 obvious.

a. Claim 38 was wrongly rejected because *Nace* in view of *Sridharan* do not teach or suggest a user interface configured to monitor and display the application values.

Claim 38 recites, "a user interface configured to monitor and display the application values; and a client application in communication with the user interface and the COBOL monitor module, the client application configured to request the program variables of the COBOL program from the COBOL monitor module and provide the program variables to the user interface for display via the user interface responsive to a request from the user interface." Appellant notes that claim 38 includes features substantially similar to those discussed above in

section VII.B.1.b. Therefore, the arguments presented in section VII.B.1.b. are repeated for this claim.

C. 35 U.S.C. § 103 Rejections – Claim 5 is rejected over *Nace* in view of *Sridharan* and in further view of *Kashima*.

1. *Nace* in view of *Sridharan* and in further view of *Kashima* do not render claim 5 obvious.

Claim 5 depends from claim 1. Therefore, the arguments presented in sections VII.B.1.a. and VII.B.1.b. are repeated for this claim. Appellant submits that *Kashima* does not cure the deficiencies of *Nace* and *Sridharan* noted above.

D. 35 U.S.C. § 103 Rejections – Claims 25, 31, and 32 are rejected over *Nace* in view of *Tao-2*.

1. *Nace* in view of *Tao-2* do not render claims 25, 31, and 32 obvious.

Claims 25, 31, and 32 depend from claim 21. Therefore, the arguments presented in sections VII.A.1.a. and VII.A.1.b. are repeated for these claims. Appellant submits that *Tao-2* does not cure the deficiencies of *Nace* noted above.

E. 35 U.S.C. § 103 Rejections – Claims 27 and 29 are rejected over *Nace* in view of *Huang*.

1. *Nace* in view of *Huang* do not render claims 27 and 29 obvious.

Claims 27 and 29 depend from claim 21. Therefore, the arguments presented in sections VII.A.1.a. and VII.A.1.b. are repeated for these claims. Appellant submits that *Huang* does not cure the deficiencies of *Nace* noted above.

F. 35 U.S.C. § 103 Rejections – Claims 33 and 34 are rejected over *Nace* in view of *Tao-2* and further in view of *Tao-1*.

1. *Nace* in view of *Tao-2* and further in view of *Tao-1* do not render claims 33 and 34 obvious.

Claims 33 and 34 depend from claim 21. Therefore, the arguments presented in sections VII.A.1.a. and VII.A.1.b. are repeated for these claims. Appellant submits that *Tao-1* and *Tao-2* do not cure the deficiencies of *Nace* noted above.

VIII. CONCLUSION

In view of the above arguments the Appellant respectfully requests that the Final Rejection of the claims be reversed and the case advanced to issue. Should the Examiner feel that a telephone interview would advance prosecution of the present application, the Appellant invites the Examiner to call the attorneys of record.

The Commissioner is hereby authorized to charge payment of any further fees associated with any of the foregoing papers submitted herewith, or to credit any overpayment thereof, to Deposit Account No. 21-0765, of Sprint Communications Company, L.P.

Respectfully submitted,
CONLEY ROSE, P.C.

Date: December 21,2009

/Michael W. Piper/
Michael W. Piper
Reg. No. 39,800

ATTORNEY FOR APPELLANT

CONLEY ROSE, P.C.
5601 Granite Parkway, Suite 750
Plano, Texas 75024
(972) 731-2288
(972) 731-2289 (facsimile)

IX. CLAIMS APPENDIX

1. (Previously Presented) A system for non-intrusively monitoring an application, comprising:

at least one application stored on a computer-readable medium, the at least one application creates a shared memory area and stores application values in the shared memory area;

a first module stored on a computer-readable medium that shares and attaches to the shared memory area that is used by the at least one application during real-time operation, the first module reads application values from the shared memory area that have been stored in the shared memory area by the at least one application during real-time operation;

a second module stored on a computer-readable medium in communication with the first module that requests the first module to read the application values, the second module receives the application values from the first module; and

a third module stored on a computer-readable medium in communication with the second module that displays the application values.

2. (Previously Presented) The system of Claim 1, wherein the shared memory area is further defined as a shared memory of the application.

3. (Previously Presented) The system of Claim 1, wherein the first module is further configured to attach to the shared memory area used by the at least one application to read the application values.

4. (Previously Presented) The system of Claim 1, wherein the application values are further defined as at least one application variable and a value for the at least one application variable.

5. (Previously Presented) The system of Claim 1, wherein the first module is further configured to communicate the application values to the second module in hypertext markup language format.

6. (Original) The system of Claim 1, wherein the third module is further defined as a graphical user interface.

7. (Previously Presented) The system of Claim 6, wherein the graphical user interface is further configured to receive an input identifying the application values to be read and configured to request the application values identified to the first module, via the second module, and wherein the first module is configured to read the requested application values data from the shared memory area and return the application variables to the graphical user interface, via the second module.

8. (Previously Presented) The system of Claim 6, wherein the graphical user interface is further configured to receive an input identifying requested application values to be displayed.

9. (Previously Presented) The system of Claim 1, wherein the first module is further configured as a socket server and wherein the second module is further configured as a socket client such that the first and second modules communicate via a socket connection.

10. (Previously Presented) The system of Claim 1, wherein the first module reads application values stored in the shared memory area by the at least one application while the at least one application is running.

11. (Previously Presented) The system of Claim 10, wherein the first module reads application values stored in the memory area by the at least one application without interfering with the operation of the at least one application.

12. (Previously Presented) A method of non-intrusively monitoring operation of an application, comprising:

running an application in a real-time manner;

creating a memory area;

generating, by the application, application values during operation of the application;

writing, by the application, the application values in the memory area during the operation of the application;

reading, by a monitor, the memory area used by the application to obtain the application values, wherein at least one of the application values is not output by the application;

and

displaying the application values read from the memory area.

13. (Previously Presented) The method of Claim 12, further comprising:

requesting, by a client, application values from the monitor; and

communicating the application variables from the monitor to the client.

14. (Previously Presented) The method of Claim 13, further comprising:

requesting application values;

running a plurality of applications in a real-time manner;

generating application values stored in one or more memory areas during operation of the plurality of applications;

reading the one or more memory areas used by the plurality of applications to obtain the application values; and

displaying the requested application values.

15. (Previously Presented) The method of Claim 14, wherein the memory area is further defined as a block of memory and wherein the monitor reads at least some of the application variables stored in the block of memory.

16. (Previously Presented) The method of Claim 13, further comprising providing a memory manager and wherein the monitor registers with the memory manager to obtain a location of the memory area used by the application to store the application values.

17. (Original) The method of Claim 13, further comprising:

generating new application values by the application stored in the memory area, at least one of the new application values defined as a new value for a variable of the application;

requesting, by the client, that the monitor re-read the application values stored in the memory area;

re-reading, by the monitor, the memory area to obtain the new application values.

18. (Previously Presented) The method of Claim 17, wherein the monitor reads the application values while the application is running.

19. (Previously Presented) The method of Claim 13, wherein the monitor is configured as a socket server and wherein the client is configured as a socket client such that the communication between the monitor and client is via a socket connection.

20. (Original) The method of Claim 12, wherein the application values are further defined as a variable of the application and a value of the variable.

21. (Previously Presented) A system for non-intrusively monitoring variables during operation of an application, comprising:

a compile listing stored on a computer-readable medium having an address map with an offset associated with each of a plurality of variables of an application; and

a module stored on a computer-readable medium that performs reading of the compile listing and obtaining the offset of at least one of the plurality of variables of the application, the module performs attaching to an address space used by the application during real-time operation to obtain a value for one or more of the plurality of variables written to the address space by the application during the real-time operation of the application using the offset.

22. (Previously Presented) The system of Claim 21, wherein the module is further configured to read the compile listing and convert at least one of the plurality of variables to the associated offset.

23. (Previously Presented) The system of Claim 21, wherein the module is further configured to search the compile listing and display the plurality of variables of the application for selection by a user.

24. (Original) The system of Claim 23, wherein the module is responsive to selection by the user of one of the plurality of variables to obtain the value for the selected one of the plurality of variables using the offset to locate the value of the variable in the address space.

25. (Previously Presented) The system of Claim 24, wherein the module is further configured to display the selected one of the plurality of variables.

26. (Original) The system of Claim 21, wherein the address space is further defined as a memory space and wherein the module attaches, using a socket layer, to the memory space used by the application.

27. (Original) The system of Claim 26, wherein the module attaches, using the offset, to the memory space used by the application via an operating system service.

28. (Previously Presented) The system of Claim 21, wherein the monitor is further configured, using the compile listing, to query the address map for one or more of the plurality of variables of the application.

29. (Original) The system of Claim 21, wherein the module is further defined as a subtask of the operating system.

30. (Previously Presented) The system of Claim 21, wherein the module is further configured to attach to the memory space where the application is operating and overwrite the value for one or more of the plurality of variables using the offset.

31. (Previously Presented) The system of Claim 21, wherein the module comprises:

a reader component configured to perform reading the compile listing and further configured to perform converting at least one of the plurality of variables of the application to the associated offset; and

a search component that performs receiving the associated offset of the at least one of the plurality of variables from the reader component, the search component configured to perform attaching to the application and further operable to locate the value of the at least one of the plurality of variables using the offset.

32. (Previously Presented) The system of Claim 21, further comprising a display component operably coupled to the module to perform receiving the value for the one or more of the plurality of variables, the display component configured to perform displaying the value.

33. (Previously Presented) The system of Claim 32, wherein the display component is configured to employ the value to display a heartbeat.

34. (Previously Presented) The system of Claim 32, wherein the display component is configured to employ the value to display as a percentage complete.

35. (Previously Presented) A system for non-intrusively monitoring COBOL application values, the system comprising:

a COBOL program stored on a computer-readable medium that creates a shared memory area through a technical layer, generates program values, and stores the program values in the shared memory area during real-time operation of the COBOL program; and

a COBOL monitor module stored on a computer-readable medium that shares the shared memory area with the COBOL program through the technical layer, and the COBOL monitor module reads the program values stored in the shared memory area by the COBOL program during real-time operation of the COBOL program.

36. (Previously Presented) The system of Claim 35, further comprising:

a second COBOL program configured to generate second program values and store the program values in the shared memory area during real-time operation of the second COBOL program, and wherein the COBOL monitor module is further configured to read the second program values stored in the shared memory area by the second COBOL program.

37. (Previously Presented) The system of Claim 35, further comprising:

a second memory area; and

a second COBOL program configured to generate second program values and store the program values in the second memory area during real-time operation of the second COBOL program, and wherein the COBOL monitor module is further configured to read the second program values stored in the second memory area by the second COBOL program.

38. (Previously Presented) The system of Claim 35, further comprising:

a user interface configured to monitor and display the application values; and

a client application in communication with the user interface and the COBOL monitor module, the client application configured to request the program variables of the COBOL program from the COBOL monitor module and provide the program variables to the user interface for display via the user interface responsive to a request from the user interface.

X. EVIDENCE APPENDIX

None.

XI. RELATED PROCEEDINGS APPENDIX

None.